主要内容

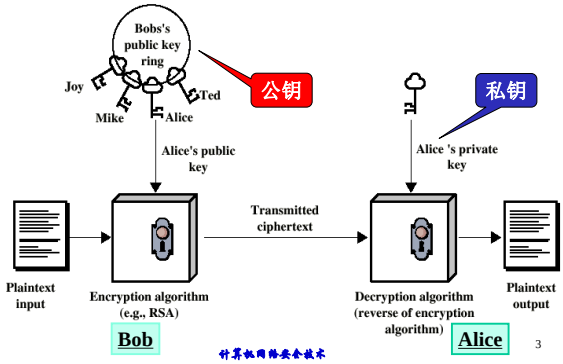
- 公钥算法的应用
- ◆认证技术
  - 消息认证基本方法
  - 身份认证技术概述
    - 基本概念
    - 口令认证的基本原理

公钥算法的应用

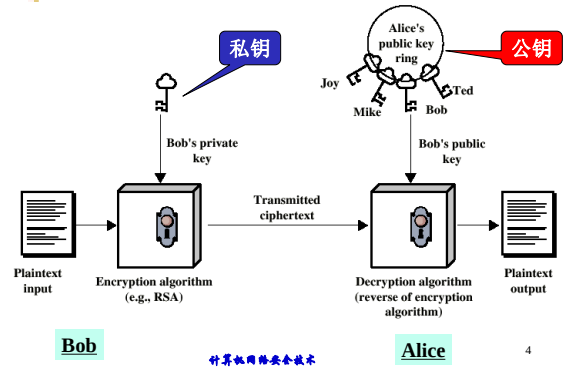
- ◆ 三类应用
  - 加/解密 Encryption/decryption (provide secrecy)
  - 数字签名 Digital signatures (provide authentication)
  - 密钥 交换 Key exchange (of session keys)

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

公钥算法应用(1)： 加/解密



公钥算法应用(2)： 数字签名



数字签名的需求

- 数字签名
  - 数字签名（digital signature）技术通过某种加密算法，在一条地址消息的尾部添加一个字符串，而收信人可以根据这个字符串验明发信人的身份，并可进行数据完整性检查。
- 需求
  - 签名是可信的：签名使文件的接收者相信签名者是慎重地在文件上签字的。
  - 签名不可伪造：签名证明是签字者而不是其他人在文件上签字。
  - 签名不可重用：签名是文件的一部分，不法之徒不可能将签名移到不同的文件上。
  - 签名的文件是不可改变的：在文件签名后，文件不能改变。
  - 签名是不可抵赖的：签名和文件是物理的东西，签名者事后不能声称他没有签过名。

数字签名的过程

基本的数字签名协议：

（1）Bob用其私钥对文件加密，从而对文件签名。

（2）Bob将签名的文件传给Alice。

（3）Alice用Bob的公钥解密文件，从而验证签名

这个协议也满足我们期待的特征：

（1）签名是可信的。当Alice用Bob的公钥验证信息时，她知道是由Bob签名的。

（2）签名是不可伪造的。只有Bob知道自己的私钥。

（3）签名是不可重用的。签名是文件的函数，并且不可能转换成另外的文件。

（4）被签名的文件是不可改变的。如果文件有任何改变，文件就不可能用Bob的公钥验证成功。

（5）签名是不可抵赖的。Alice不用Bob的帮助就能验证Bob的签名.

数字信封技术：混合方案

- 公开密钥密码系统的优点
  - 密钥管理方便（具有*n*个用户的网络仅需要2*n*个密钥）
  - 便于实现数字签名，适合电子商务等应用需要。
- 问题:
  - 采用公钥密码算法对长文件签名效率太低
  - 计算非常复杂，加密/解密速度远赶不上对称密钥加密系统。
- 解决方案
  - 数字签名协议和单向散列函数一起使用。
  - **数字信封技术（Digital Envelopes）**

数字签名实现中的问题

问题：文件是否需要加密?

Bob并不对整个文件签名，只对文件的散列值签名。在这个协议中，单向散列函数和数字签名算法是事先就协商好了的。

（1）Bob产生文件的散列值。

（2）Bob用自己的私钥对散列值加密，表示对文件签名。

（3）Bob将文件和散列签名送给Alice。

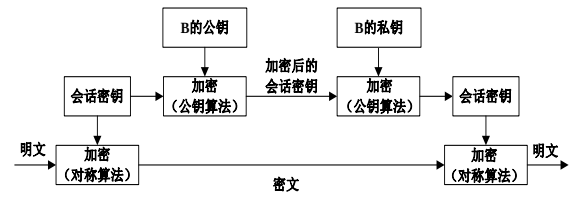
（4）Alice用Bob发送的文件产生文件的散列值，然后用Bob的公钥对签名的散列值解密。如果解密的散列值与自己产生的散列值相同，签名就是有效的。

公钥算法应用(3)：密钥交换

问题：在通信中如何交换对称密钥?

- 发送方生成一个加密数据的**会话密钥**，并用接收方的**公开密钥**对**会话密钥**进行加密，然后通过网络传输到接收方。
- 发送方对需要传输的文件用**会话密钥**进行加密，然后通过网络把加密后的文件传输到接收方。
- 接收方用自己的私钥对发送方加过密的会话密钥进行解密后得到加密文件的会话密钥。
- 接收方用会话密钥对发送方加过密的文件进行解密得到文件的明文形式。

加密与密钥交换过程



这种混合加密方式可以较好地解决**加密/解密运算的速度**问题和**密钥分配管理**问题。

密钥生成工具

- 在linux中生成密钥（公钥和私钥对）
  - **OpenSSH和ssh-keygen**
    - OpenSSH是SSH协议的开源版本（SSH：Secure SHell），提供了实现SSH协议的很多工具。其中ssh-keygen用于生成，管理和转换用于认证的密钥和证书。
  - **OpenSSL**
    - 是一个强大的安全套接字层密码库，包括了加密算法，常用密钥和证书管理，SSL协议等功能。（openssl genrsa）
  - **Keytool工具管理证书**
    - Java提供的密钥、证书和证书库管理工具。可以完成生成密钥，生成证书等各种操作。
  - **GPG**
  - 。。。

应用工具：GPG

GnuPG

- GPG即GNU Privacy Guard，简称GPG，开源的加密和数字签名工具
- GPG vs. PGP
  - 1991年，程序员Phil Zimmermann开发加密软件PGP（Pretty Good Privacy），商业软件
  - GnuPG(*www.gnupg.org*)：是PGP的非商业化版本，用于对Email、文件及其他数据的收发进行加密与数字签名，确保通信数据的可靠性和真实性
  - openpgp已提交IETF标准化

- **OpenPGP** 是一种加密标准和协议（RFC4880）。
- **PGP**（Pretty Good Privacy）是这个标准的商业软件。
- **GPG**（GnuPG）是这个标准的开源实现。

实现一个文件加解密程序

可作为大作业的选题

1. 产生你的密钥（公钥和私钥）
2. 密钥管理
3. 加/解密一个文件
4. 签名/验证一个文件

问题：

- **你发布的公钥如何可信？**
- **如何认证发送方（不可抵赖）？**

随机数

- 生成密钥
  - 用于对称密钥和消息认证码
- 生成密钥对
  - 用于公钥密码和数字签名
- 生成初始化向量（IV）
  - 用于分组密码的CBC、CFB和OFB模式
- 生成临时握手信号nonce
  - 用于防御重放攻击，以及分组密码的CTR模式
  - 生成salt
    - 用户基于口令的密码等

随机数的性质

- 随机性
  - 均匀分布：每个数出现的频率大致相等
  - 独立性：无法由其他数推导出来
- 不可预测性
- 不可重现性

	随机性	不可预测性	不可重现性
弱伪随机数	√	×	×
强伪随机数	√	√	×
真随机数	√	√	√

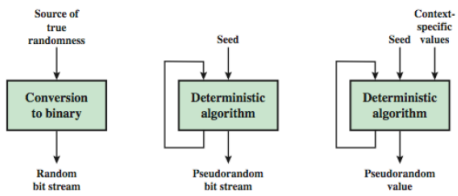
伪随机数生成

- 线性同余法可作为产生均匀分布的伪随机数的算法
- 伪随机数生成器(RNG)
  - 线性同余法

$$x_n = ax_{n-1} + b \bmod m$$

    - C语言库函数rand，Java的java.util.Random类等
    - 问题：不具备不可预测性，不能用于密码技术（由 x_i解出a，b，m）
  - 单向散列函数
  - 密码法：DES或RSA等
  - ANSI X9.17： 3DES

随机和伪随机数发生器



几点说明

- 计算机产生的伪随机数既是随机的又是有规律的。
- 随机种子可来自系统时钟，即来自计算机主板上的定时/计数器在内存中的记数值。
- 随机数是由随机种子根据一定的计算方法计算出来的数值。所以，只要计算方法一定，随机种子一定，那么产生的随机数就不会变。
- 如果想在一个程序中生成随机数序列，需要至多在生成随机数之前设置一次随机种子。

对伪随机数的攻击

- 由于伪随机数可用来生成密钥，经常成为攻击者的目标
  - **对种子进行攻击**：如果攻击者知道了伪随机数的种子，那么他就能够知道这个伪随机数所生成的全部伪随机数列。
  - **对随机数池进行攻击**：随机数池的内容如果被攻击者知道，伪随机数的种子就有可能被预测出来。

消息认证的基本方法

Message Authentication

消息认证

- 信息完整性（I）
  - 信息在存储和传输过程中不被非法篡改、破坏、增删，能真实无误的到达目的地的特性
- 在网络通信中，有一些针对消息内容的攻击方法
  - 伪造消息
  - 篡改消息内容
  - 改变消息顺序
  - 消息重放或者延迟
- 消息认证：对收到的消息进行验证，证明确实是来自声称的发送方（可信），并且没有被修改过
  - 如果在消息中加入时间及顺序信息，则可以完成对时间和顺序的认证
    - **消息的时效性**

非加密的消息认证

是否需要加密整个消息?

- 对称加密方法的问题
  - 例如：分组加密算法ECB模式下，攻击者重排密文分组次序后仍可以被解密
- 无需保密的消息认证，例如
  - 消息广播
  - 重负载的主机
  - 计算机程序分发：认证

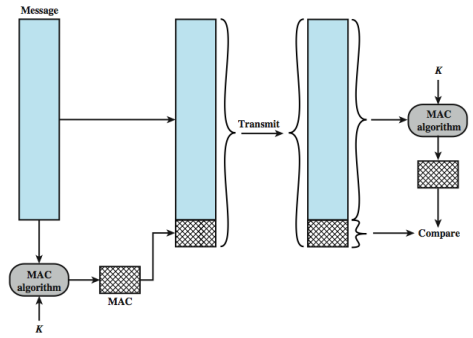
消息认证的三种方式

- 消息加密Message encryption：即用整个消息的密文作为认证标识
  - 接收方必须能够识别错误（检错码）
- 消息认证码 MAC：即采用一个公开函数，加上一个密钥产生一个固定长度的值作为认证标识
- 散列函数Hash function：即一个公开函数将任意长度的消息映射到一个固定长度的散列值，作为认证标识

Message Authentication Code (MAC)

- 消息认证码MAC：使用一个双方共享的秘密密钥生成一个固定大小的小数据块，并加入到消息中，称MAC，或密码校验和(cryptographic checksum)
  - 用户A和用户B，共享密钥K，对于消息M，$MAC=C_K(M)$
- 如果接收方计算的MAC与收到的MAC匹配，则
  - 接收者可以确信消息M未被改变
  - 接收者可以确信消息来自所声称的发送者
  - 如果消息中含有序列号，则可以保证正确的消息顺序
- MAC函数类似于加密函数，但不需要可逆性。因此在数学上比加密算法被攻击的弱点要少

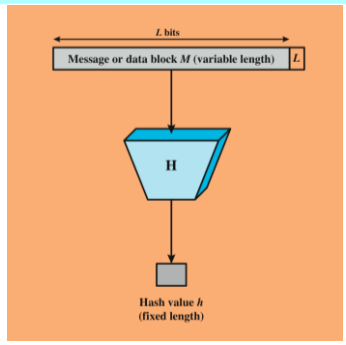
利用消息认证码MAC进行消息认证



消息摘要和Hash函数

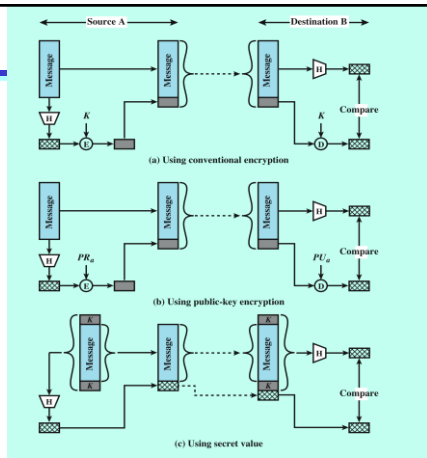
- ◆单向散列函数（Hash函数，哈希函数，散列函数）
  - ❖MAC的一种替代方法
- ◆特点
  - ❖是把可变长输入数据转换成固定长度输出数据的函数
  - ❖这个定长的输出数据称为输入数据的消息摘要Digest，也称为散列值、哈希值或数字指纹
- ◆消息摘要是保证信息完整性的基本技术之一
  - ❖Hash函数
  - ❖两个典型的Hash算法：MD5 和SHA-1
  - ❖消息摘要的生成和验证过程

Hash函数



利用单向散列函数进行消息认证

- ◆对称加密
  - ❖ 接收方和发送方共享加密密钥
  - ❖ 例如：使用DES算法生成认证码
- ◆公钥加密
  - ❖ 数字签名和认证
- ◆秘密值加密
  - ❖利用散列函数，但不使用加密的消息认证技术
  - ❖秘密密钥本身不被发送



安全散列（hash）函数的特性

- 接受的输入数据没有长度限制：对输入任何长度的数据能够生成该输入消息固定长度的输出
- 快速性：已知m，计算hash(m)是容易的
- 单向性（抗原象）：已知哈希函数的输出，要求它的输入是困难的，即已知c=hash(m)，求m是困难的
- 抗碰撞性：已知hash(m_1)=c_1，构造m_2使hash(m_2)=c_1是困难的
- 雪崩性：c=hash(m)，c的每一比特都与m的每一比特有关，并有高度敏感性。即每改变m的一个比特，都将对c产生明显影响

散列函数的安全性

- 对安全散列函数的攻击有两类
  - 密码分析
  - 蛮力攻击
- 攻击者的目标通常是找到两个不同消息映射为同一值
  - 穷举攻击法(Exhaustive Attack)
  - 生日攻击(Birthday Attack)
- 应用
  - 口令*Passwords*: 操作系统中存储口令的哈希值
  - 入侵检测*intrusion detection*: 对系统中每个文件存储哈希值 H(F)

生日攻击

- 依赖于消息摘要的长度，即Hash值的长度
- 生日悖论：在一个教室中，找一个与某人生日相同的概率不小于0.5时，所需学生为183人。但要一个教室中至少有两个学生的生日在同一天的概率不小于0.5的学生人数仅为23人。
  - 这是因为对于第一种情况，与某个已知生日的人同日生的概率为1/365。若教室中有t人，则至少找到一人与此人同日生的概率为p=1-(364/365)$^{t-1}$。易于解出，当t≥183时可使p>0.5。
  - 对于第二种情况，第一个人在特定日统计生的概率为1/365，而第二人不在该日生的概率为(1-1/365)，类似地第三人与前两位不同日生的概率为(1-2/365)，以此类推，t个人都不同时生日概率为(1-1/365)(1-2/365)...(1-(t-1)/365)，因此，至少有两人于同日生的概率为1-(1-1/365)(1-2/365)...(1-(t-1)/365)，解之当，t≥23时，p>0.5。
  - 对于n比特Hash值的生日攻击，由上式可计算出，当进行$2^{n/2}$次的选择明文攻击下成功的概率将超过0.63.

针对生日攻击的安全条件

- 消息摘要必须足够的长
  - 一个40比特长的消息摘要是很不安全的，因为仅仅用2^{20} (大约一百万)次随机Hash可至少以1/2的概率找到一个碰撞。
  - 通常建议消息摘要的长度至少应选取为128比特，此时生日攻击需要约2^{64}次Hash。
    - 统计结果表明，如hash(m)的长度为128位(bit)时，则任意两个分别为M_1 M_2的输入报文具有完全相同的h(m)的概率接近于零
  - MD5标准的输出长度选为128比特
  - SHA-1标准的输出长度选为160比特

一个最简单的Hash函数

- 将输入数据分成若干等长的分组
- 对每个分组按位进行异或（XOR）运算
- 结果便为其消息摘要

	Bit 1	Bit 2	• • •	Bit n
Block 1	b_{11}	b_{21}		b_{n1}
Block 2	b_{12}	b_{22}		b_{n2}
	• • •	• • •	• • •	• • •
Block m	b_{1m}	b_{2m}		b_{nm}
Hash code	C_1	C_2		C_n

SHA-1(Secure hash Algorithm)算法

- 1993年，NIST (National Institute of Standards Technology，美国国家标准技术局)开发
- 1995年，发布改进版本FIPS PUB 181，SHA-1，Internet RFC3174
- SHA-1算法允许的最大输入报文的长度不超过2^{64}位，输出160比特的报文摘要。
  - SHA算法计算时是按照512位的分组进行处理的
- SHA-2：2001年发布，包括SHA-224、SHA-256、SHA-384、SHA-512、SHA-512/224、SHA-512/256。
- SHA-3：2015年正式发布，可替换的加密散列算法。

SHA

算法和变体		输出散列值长度（bits）	中继散列值长度（bits）	数据区块长度（bits）	最大输入消息长度（bits）	循环次数	使用到的运算符	碰撞攻击（bits）	性能示例[3]（MiB/s）
MD5（作为参考）		128	128 (4 × 32)	512	无限[4]	64	And, Xor, Rot, Add (mod 2^{32}), Or	<64（发现碰撞）	335
SHA-0		160	160 (5 × 32)	512	$2^{64}-1$	80	And, Xor, Rot, Add (mod 2^{32}), Or	<80（发现碰撞）	-
SHA-1		160	160 (5 × 32)	512	$2^{64}-1$	80		<80[5]（发现碰撞[6]）	192
SHA-2	SHA-224	224	256 (8 × 32)	512	$2^{64}-1$	64	And, Xor, Rot, Add (mod 2^{32}), Or, Shr	112	139
	SHA-256	256						128	
	SHA-384	384	512 (8 × 64)	1024	$2^{128}-1$	80	And, Xor, Rot, Add (mod 2^{64}), Or, Shr	192	154
	SHA-512	512						256	
	SHA-512/224	224						112	
	SHA-512/256	256						128	

SHA-1的应用

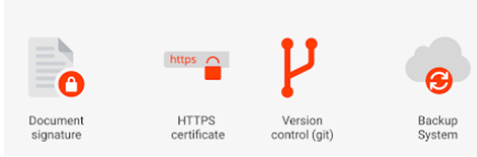
- SHA-1在许多安全协议中广为使用
  - 包括TLS和SSL、PGP、SSH、S/MIME和IPsec
  - 曾被视为是MD5（更早之前被广为使用的散列函数）的后继者。
  - 但SHA-1的安全性在2000年以后已经不被大多数的加密场景所接受。

SHA-1碰撞

- 2013年，荷兰 CWI 研究机构的马克·斯蒂文斯 (Marc Stevens)曾发表一篇论文，专门介绍了创建SHA-1碰撞的理论性方法
  - 首先创建了一份专门作的PDF前缀，用以生成两份拥有任意不同内容的文档，但二者同时具备相同的SHA-1摘要
- 2017年2月23日，荷兰密码学研究小组CWI和Google正式宣布攻破了SHA-1。SHA1碰撞攻击称之为“SHAttered”攻击
  - 谷歌发布了两个具有相同SHA-1哈希值但内容不同的PDF文件[PDF1，PDF2]。
  - “SHAttered”攻击比暴力破解攻击速度快10万倍，在亚马逊的云计算平台上执行成本仅为11万美元。

受攻击影响的系统

- SHA1算法仍被广泛使用。任何依赖SHA-1进行数字签名、文件完整性或文件识别的应用程序都可能受“SHAttered”攻击影响。
  - 数字证书签名、电子邮件PGP / GPG签名
  - 软件供应商签名、软件更新
  - ISO校验和备份系统、重复数据删除系统、GIT等等。
- 补救措施：替换成更安全的算法，如SHA-256和SHA-3



SHA-1算法处理过程

- 信息填充
  - 首先需要对信息进行填充，使其字节长度对512求余的结果等于448，即填充后的消息比512的整数倍 少64位。
  - 用64位表示填充前的报文长度，附加在填充后的结果后面。
- 初始化消息摘要(MD)缓存区
  - SHA-1使用160位的缓存来存放算法的中间结果和最终的散列值。这个缓存由5个32位的寄存器A, B, C,D,E构成。
- SHA-1将寄存器的初始值设为

  A=0x67452301 B=Oxefcdab89
  C=Ox98badcfe D=0x10325476 E=Oxc3d2elf0

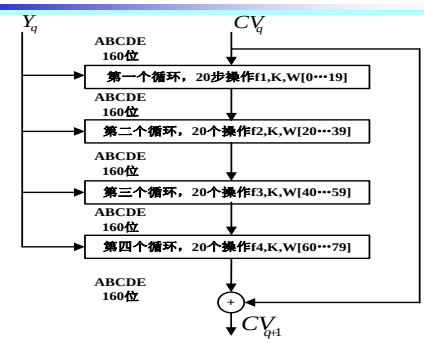
SHA-1算法处理过程

初始化消息摘要(MD)缓存器：

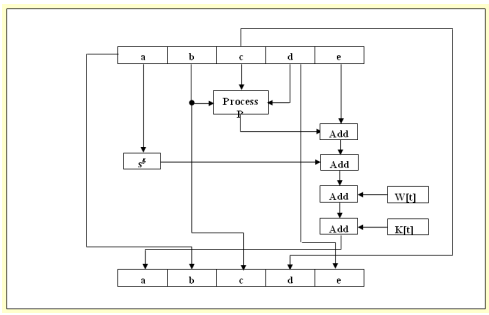
处理算法的核心是一个4个循环的压缩函数模块，其中每个循环由20个处理步骤组成。

- 在每个循环中使用不同的原始逻辑函数，分别表示为fl, f2, f3 , f4。
- 每个循环都对512位的分组 Y_q进行处理，并对160位的缓存器A, B, C, D, E进行更新。循环中还需要使用一个额外的常数K。
- 当4个循环都完成后，缓存器A, B, C, D, E中的结果与 Y_q的输入链接变量的初值(CV_q)相加，就可以得到下一个分组 Y_{q+1}的输入链接变量CV_{q+1}
- 当所有L个512位的分组都处理完成后，最后第L个阶段产生的输出CVL就是160位的报文摘要，结果保存在缓存器A,B,C,D,E中。

SHA-1算法处理过程



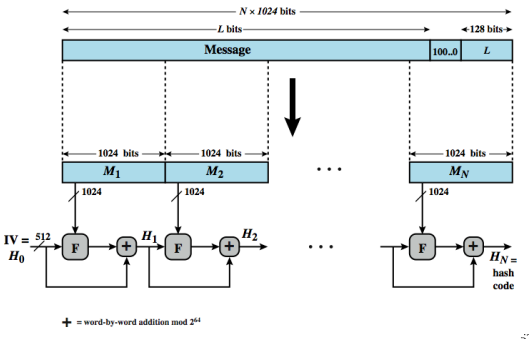
SHA-1处理过程



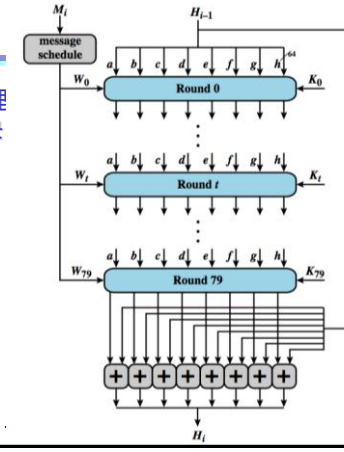
SHA-512

- 该算法输入消息的最大长度2^{128}位
- 信息填充
  - 填充消息使其长度与896模1024同余
  - 用128位表示填充前的报文长度，附加在填充后的结果后面
- 初始化消息摘要(MD)缓存区
  - SHA-512使用512位的缓存来存放算法的中间结果和最终的散列值。这个缓存由8个64位的寄存器A, B, C,D,E,F,G,H构成。
- 以1024位（128-word）为单位处理消息
  - 80轮运算：循环移位；与（AND）、或（OR）、非（NOT）和异或（XOR）的逻辑运算
- 输出512位的消息摘要

SHA-512



SHA-512 轮次：处理单个1024位的数据块



SHA-3

- SHA-2 及其后继版本具有相同的结构和相同的数学操作。
- 2007年，NIST征集下一代散列函数，能取代SHA-2
- 2012年发布新的散列函数 SHA-3

需求

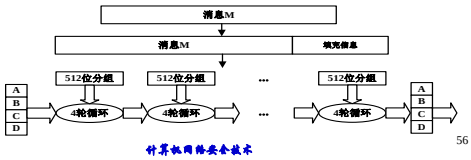
- **支持散列值的长度为：224, 256,384, 512 位**
- **算法能够一次处理较小的消息分组（512或1024位），而不必把整个未处理的消息放到缓存中**

MD5算法

- MD5：Message-Digest Algorithm 5
  - 1991年由MIT Laboratory for Computer Science（IT计算机科学实验室）和RSA Data Security Inc（RSA数据安全公司）的Ronald L. Rivest教授开发，经MD2、MD3和MD4发展而来
- MD5将任意长度的输入消息变换成一个128bit的大整数，并且它是一个不可逆的变换算法
- 应用
  - MD5的典型应用是对一段信息（Message）产生信息摘要（Message-Digest），以防止被篡改
  - MD5还广泛用于加密技术上。比如在UNIX系统中用户的密码就是以MD5（或其它类似的算法）经加密后存储在文件系统中
    - 当用户登录的时候，系统把用户输入的密码计算成md5值，然后再去和保存在文件系统中的md5值进行比较，进而确定输入的密码是否正确。

MD5算法处理过程

- MD5以512位分组来处理输入的信息，
- 每一分组又被划分为16个32位子分组，
- 每一个分组要经过4轮循环处理，
- 处理前以A、B、C、D这4个缓冲区中的值作为输入参数，处理后输出四个32位值，A、B、C、D分别加上这4个32位值后再作为下一个分组处理所用的输入参数。
- 所有分组经过了4轮主循环的处理后，输出四个32位最终值，将这四个32位值级联后即生成了一个128位摘要值



信息填充

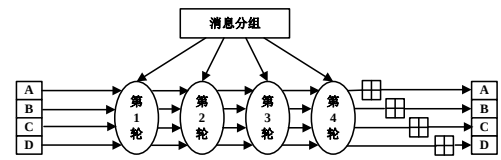
- 在MD5算法中，首先需要对信息进行填充，使其字节长度对512求余的结果等于448
- 方法：
  - 在信息的后面填充一个1和若干个0
  - 附加一个64位的二进制数据，此数据表示填充前信息的长度
  - 要散列的数据

循环处理

- 填充后的消息被划分成若干个512位的分组，每一分组又被划分为16个32位子分组。这些分组作为算法的输入信息进行处理。
- 初始化链接变量（Chaining Variable）
  - MD5中有A、B、C、D四个缓冲区（32位的寄存器）中，最开始存放四个被称作链接变量的32位的整数参数，这些参数将用于下一步的摘要算法中。这四个缓冲区分别被初始化为以下的值：
  - A=0x01234567　　B=0x89abcdef
    C=0xfedcba98　C=0x76543210
- 当设置好这四个链接变量后，将它们复制到另外四个变量中：A到a，B到b，C到c，D到d。abcd组成128位寄存器，用于保存中间结果和最终结果

循环处理

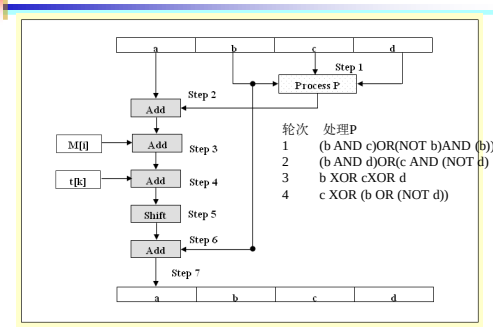
- 消息的每一个分组均经过四轮循环处理，处理前将缓冲区A、B、C、D的内容作为输入参数，处理后的结果再加入到A、B、C、D中去。



循环处理

- 每一轮循环对一个分组进行16次操作。
- 每次操作：
  - 对a, b, c和d中的其中三个作一次非线性函数运算
  - 然后将所得结果加上第四个变量、分组的一个子分组和一个常数。
  - 再将所得结果循环左移，并加上a, b,c或d中之一。
  - 最后用该结果取代a, b, c或d中之一

MD5处理过程



a=b+((a+Process P(b,c,d)+M[i]+T[k])<<<s)

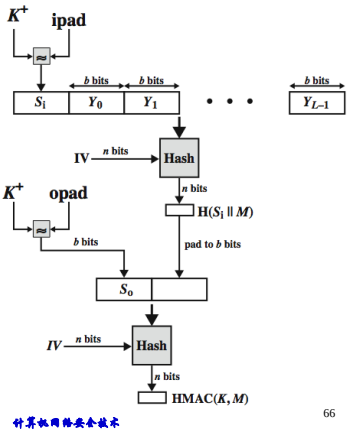
HMAC（Hash-MAC）

- 消息认证码HMAC
  - 密钥与散列算法相结合，*RFC2104*
    - 散列算法（哈希函数）不限于一种
  - 在IPSec，TLS，SET中实现
    - HMAC-SHA-1, HMAC-MD5, HMAC-RIPEMD
- 方法

  $HMAC_K(M)$= Hash[$(K^+$ XOR opad) ||
  Hash[$(K^+$ XOR ipad) || M)]]
  - K^+ ： 填充后的密钥
  - opad, ipad ： 特定比特序列01011100（ox5C），00110110（0x36）
  - Hash：嵌入的散列函数，如SHA

HMAC的步骤

- 密钥填充
- 填充后的密钥与ipad进行XOR
- 与消息组合
- 计算散列值
- 填充后的密钥与opad进行XOR
- 与散列值组合
- 计算散列值



HMAC的安全性

- 安全性依赖于嵌入的散列函数的强度
  - 攻击者根据随机的秘密IV值计算压缩函数的输出
    - 暴力破解$O(2^n)$, 或生日攻击
  - 即使IV是随机的、秘密的，攻击者也可以找到散列函数的碰撞
    - 例如，找到 M 和 M ‘，使得 H(M) = H(M')
    - 生日攻击 $O(2^{n/2})$

三、认证技术

认证服务

- 认证目的
  - 证实**收到信息**的真实性
  - 证实**存储数据**的真实性
  - 证实**发送信息**的真实性
  - 证实**信息**的时效 性
  - 证实**对方**的真实性
  - 使**接收方**的活动可审计与仲裁
  - 提供**第三方公证**的可能性

信息安全的需求：CIA模型

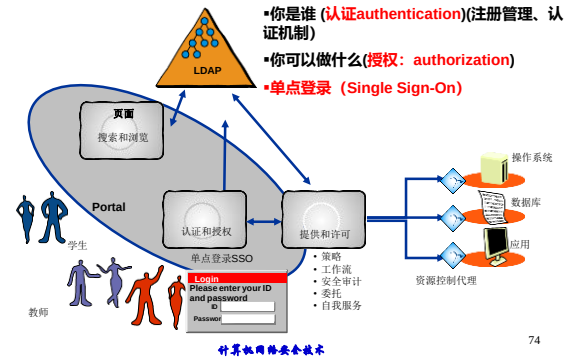
- 保密性Confidentiality
- 完整性Integrity
  - 系统完整性
  - 数据完整性
  - 真实性 authenticity
    - 认证
      - **消息认证**
      - **身份认证：验证真实身份和所声称身份相符的过程**
- 可用性Availability
- ……

身份认证

- 身份认证涉及两方面内容：
  - 识别、明确访问者身份
  - 验证对访问者所说的身份进行确认
- 个人身份验证方法
  - 验证他知道什么
  - 验证他拥有什么
  - 验证他的生物特征
    - 静态：生理特征
    - 动态：行为特征

身份认证的例子

用户身份管理、访问管理、单点登录



基于口令的认证

- 应用最广泛的防御手段
  - 用户提供 用户名name/login和口令 password
  - 系统将该口令和以前保存的该用户ID的口令进行比较
- 用户 ID:
  - 决定用户是否被授权访问系统
  - 决定用户的访问权限 privileges
  - 用于访问控制 access control

“坏”口令和“好”口令

- “坏”口令
  - frank
  - Fido
  - password
  - 4444
  - Pikachu
  - 102560
  - AustinStamp
- “好”口令?
  - jflej,43j-EmmL+y
  - 09864376537263
  - P0kem0N
  - FSa7Yago
  - 0nceuP0nAt1m8
  - PokeGCTall150

口令的脆弱性

- 离线字典攻击
  - 攻击者获得口令文件的访问权，并将其中的口令散列值与常用口令的散列值进行比较。
- 特定账户攻击
  - 针对特定的用户不断猜测口令，直到发现正确的口令
- 常用口令攻击
- 单用户口令猜测
- 工作站劫持
- 利用用户疏漏，社会工程学策略
- 口令重复利用
- 电子监视

防御措施

- 防止非授权访问口令文件
- 使用入侵检测技术进行攻击行为识别
- 对不安全口令的快速恢复
- 用户锁定机制：限制登录次数
- 禁止用户使用弱口令（常用口令）
- 加强口令保护策略，包括口令最小长度，字符集，禁止使用用户ID，更换口令时间等
- 在工作中处于非活动状态时自动注销
- 禁止对网络设备使用相似口令

散列口令的使用

- 口令安全技术广泛使用散列函数和“salt”值
  - 与时间相关的取值
  - 随机数或伪随机数
- 生成随机salt值s，加入到口令中计算哈希值

  y = h(password, s)

  在口令文件中存储 (s,y)
- 注意： salt值 s不是秘密的

“盐” salt的作用

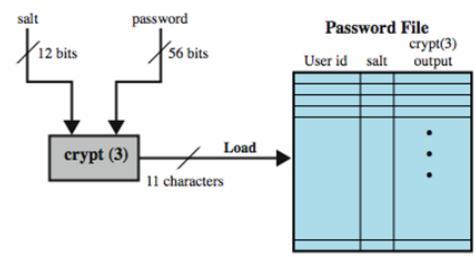
- 防止在口令文件中发现重复口令
  - 攻击者难以发现一个用户是否在不同系统中使用了相同的口令。
- 增加离线字典攻击的难度
  - 攻击者必须将其字典中的每个口令先添加salt值，再逐个计算每个用户的口令哈希值
  - 例如，salt值长度为b 位，可能的口令数量就会增加2^b倍
  - 破解口令的工作量大大增加!

UNIX的口令方案

- 最初版本
  - 8个可打印字符
  - 12位 salt值，使用修改的 DES 算法实现单向散列函数
  - 输入一个64位全0数据块，加密过程重复25次
  - 64位输出转换成11个字符序列
- 目前被认为是不适用的
  - 字典攻击
  - 用于兼容已有的财务管理软件，或在多厂商环境下使用

UNIX的口令方案

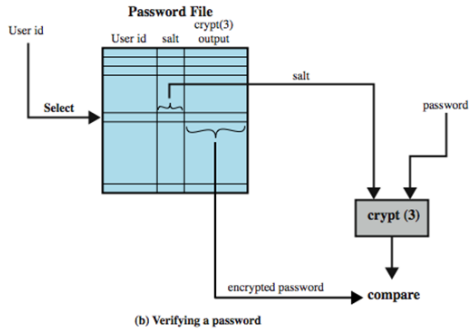
加密新口令



(a) Loading a new password

UNIX的口令方案

验证口令



(b) Verifying a password

改进实现

- 更强大的散列/ salt值方案适用于UNIX操作系统
- 推荐基于 MD5的散列函数
  - salt 值达到48位
  - 口令长度无限制
  - 产生128 位散列值
  - 使用1000次迭代的内部循环，实现减速slowdown
- OpenBSD 使用基于Blowfish对称分组密码的散列函数，称为 Bcrypt
  - Bcrypt设置口令长度最多55个字符
  - 使用 128位 salt值，产生 192 位散列值
  - 代价变量cost variable，与运算时间相关
    - The cost assigned to a new password is configurable, so that administrators can assign a higher cost to privileged users.

口令破解 Password Cracking

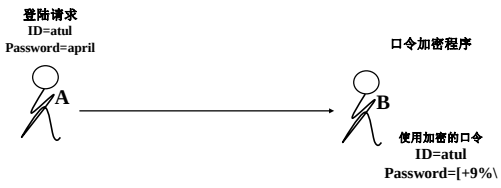
- 字典攻击 dictionary attacks
  - 开发庞大的口令字典，尝试每个口令
  - 每个口令用salt值进行散列计算，与存储的散列值进行比较
- 彩虹表攻击 rainbow table attacks
  - 对所有salt值预计算散列值，以空间代价换取时间代价
  - 庞大的散列值表（彩虹表 rainbow table）
  - 设置足够大的salt值和足够长的散列值对抗这种攻击
- 用户自行选择弱口令问题

远程用户的口令认证

- 最简单的用户认证是本地认证
  - 直接在文件中存储口令明文?
  - 加密存储口令?
    - 对称密钥加密
    - 哈希值
- 远程用户认证
  - 通过Internet、网络、通信线路等远程认证用户
    - 攻击：口令窃听，重放攻击等

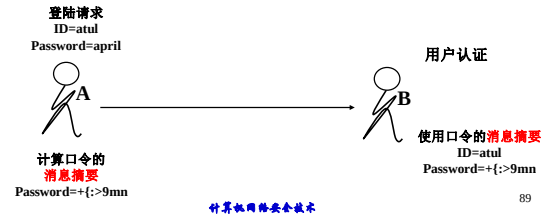
完善口令认证(一)

- 假设A是客户端，B是服务器端
  - 数据库中不能存储明文口令
- 对称密钥加密方法
  - 如何保证密钥的安全性?



完善口令认证(二)

- 使用口令的散列值（哈希值）
  - 每次对同一口令执行算法时，应得到相同的输出
  - 算法输出（即口令推导形式）不能看出原口令
  - 攻击者不可能提供错误口令而得到正确的口令推导形式



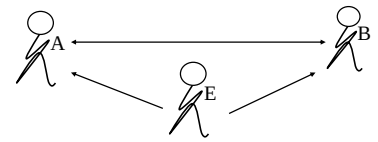
中间人攻击

中间人攻击(MITM, man in the middle)

- 使用广泛，成为对网银、网游、网上交易等最有威胁并且最具破坏性的一种攻击方式
- 如果通信双方没有任何先决条件，那么这种攻击总是存在的
- A和B的协商过程很容易受到这一类攻击
- 对策：
  - 增加A和B之间的先决知识

重放攻击

重放攻击(replay attacks)



- 偷听者可以记录下当前的通信流量，以后在适当的时候重发给通信的某一方，达到欺骗的目的
- 对策
  - 保证通信的唯一性
  - 增加时间戳

作业

1. 什么是数字信封技术?
2. 列举消息认证的三种方法
3. 比较数字签名和消息认证码这两种安全服务
4. 用户身份认证的基本方法?

大作业要求：分2次提交

分组：
- 自由组合进行分组，每组2人
- 参考给定范围选择，题目自拟

提交1：小组提交开题报告：
- 每组提交大作业开题报告（提交word文件和ppt文件，）
- 课上讨论（讲5分钟）
- 主要内容包括：选题背景和意义；相关技术分析；主要研究内容；技术路线实现方法；参考资料；小组成员分工

提交2：小组提交程序和ppt（压缩成rar或zip格式）
- 每组提交一份程序和数据，程序加注释和运行环境说明。
- 每组完成一份ppt（讲8-10分钟）
- 每人根据大作业承担的工作独立提交大作业技术报告

具体提交时间安排将在课程中心网站上发布

大作业要求：选题

（1）防火墙应用

（2）基于OpenSSL的安全传输系统

（3）Web网站漏洞扫描

（4）无线路由器的漏洞与安全

（5）网络流量分析

（6）其他感兴趣的网络安全问题

参考

- “校园网安全防范与渗透测试”
- “无线路由器漏洞与安全”
- “基于netfilter/iptables的流量限速模块”
- “基于NetfilterIPTables框架的应用层包过滤防火墙”
- “Web网站漏洞扫描”
- “安全文件传输工具”